DOCUMENT RESUME

ED 043 235 EM 008 480

AUTHOR Dwyer, T. A.
TITLE Teacher-Student Authored CAI Using the NEWBASIC/CATALYST System.
INSTITUTION Pittsburgh Univ., Pa.
SPONS AGENCY National Science Foundation, Washington, D.C.
PUB DATE [70]
NOTE 21p.

EDRS PRICE EDRS Price MF-$0.25 HC-$1.15
DESCRIPTORS *Computer Assisted Instruction, Interaction, Programing, *Programing Languages, *Secondary Education, Student Developed Materials, Teacher Developed Materials
IDENTIFIERS BASIC, *NEWBASIC CATALYST

ABSTRACT

Using an interactive computer system called NEWBASIC/CATALYST, both students and teachers can act as authors of programs. NEWBASIC/CATALYST incorporates an implementation of BASIC, system-level interactive features, and a general capability for extension through use-oriented function attachment. Interacting at the system level, students can mix the advantages of independent or "solo" mode computing with those of guided or "dual" mode interaction. Illustrations of this are given. Preliminary experience with the system was in an urban secondary school setting. (Author/MF)

Teacher-Student Authored CAI Using the NEWBASIC/CATALYST System

T. A. Dwyer

University of Pittsburgh, Pittsburgh, Pa.

ABSTRACT: The pedagogical advantages of a general purpose interactive system called NEWBASIC/CATALYST are discussed. NEWBASIC/CATALYST incorporates an advanced implementation of BASIC, system-level interactive features, and a general capability for extension through user-oriented function attachment. Application of this last feature to providing a flexible CAI scan capability is illustrated. Two examples of interaction at the system level are given, showing how students can mix the advantages of independent or "solo" mode computing with those of guided or "dual" mode interaction. Preliminary experience with the system in an urban secondary school setting is discussed.

KEY WORDS AND PHRASES: CAI, BASIC, CATALYST, NEWBASIC, education, computers in education, extended languages, interactive systems.

CR CATEGORIES: 1.50, 1.51, 3.32, 4.22.

REFERENCES:

1. Dwyer, T. A. Project Solo: A Statement of Position Regarding CAI and Creativity. Interface 4, 1 (Feb. 1970), 13-15.
2. Blanc, R. CATALYST: A Computer Assisted Teaching and Learning System for a General Purpose Time-Sharing System. M.S. Thesis. University of Pittsburgh (1968).
3. Dwyer, T. A. A Lesson Designers Guide to CATALYST/PIL. University of Pittsburgh (1969).
4. Badger, G. PIL/L: Pitt Interpretive Language for the IBM System/360 Model 50. University of Pittsburgh (revised Feb. 1969).

Teacher-Student Authored CAI Using the NEWBASIC/CATALYST System

T. A. Dwyer

University of Pittsburgh

1. INTRODUCTION

The term CAI has come to have a fairly broad meaning for most educational workers, including the use of computers for drill and practice, tutorial sessions, simulation, and structured information retrieval. There is usually the assumption, however, that the work of one person (the "author") lies behind a program with which a second person (the "student") interacts. For most systems there is the additional unspoken assumption that CAI authors are specialists who are distinct from the students and teachers who use their product. There have been CAI projects where teachers have been released to help author programs, but this almost always implies that these teachers remove themselves from actual classroom experience during such release.

An additional restriction on authors is that the style, format, and teaching strategy they employ is effectively prescribed by the bigger system within which they work. A quick look at the set of reference manuals that accompany a specialized CAI system makes it clear that there are distinct functions within such systems. In order that such functions may operate with separate staffs, con-

straints must be imposed on each group, with the author function most likely to receive the brunt of these limitations.

It has been our experience that there is considerable negative reaction to all of this on the part of many educators, simply because they will have very little control of such systems at the classroom level, and their students will have none at all. This last indictment is a serious one for the upper grade levels; today's high school youth, for example, are very much concerned with the curricula in which they are to function. The knowledgeable school administrator is further confounded by the desire to introduce real "hands-on" computing as a powerful adjunct to much of the curriculum, but at the apparent cost of duplicate systems. He knows that the drill and practice routines that might fascinate the freshman could easily become anathema to the "sopho"-more, and that he really needs the power of a multi-faceted computer utility if technology is to match the complexity of the real school world.

The purpose of this paper is to report on what appears to be a viable solution to these problems. The approach described has been in use for over a year with small test groups. It is now undergoing further development within a large urban school system (Pittsburgh) as part of an experiment* in the regional use of computers for secondary schools.

---

*Supported in part by NSF grants GJ 515 and GJ 1077.

2. SOME DISTINCTIONS

When talking about computers and learning,** we have found it useful to distinguish between what we call DUAL mode and SOLO mode usage (a terminology borrowed from the world of flight instruction). We use the terms to indicate the presence or non-presence of a pedagogically intended master program with which a student interacts. Thus we would classify a simulation as DUAL mode, even though the pedagogical intent of the master program is certainly at a more subtle level than that of a response-sensitive drill and practice routine. We would classify the writing, debugging, and revision of an original BASIC program by a student as SOLO mode, since the programs with which he interacts (e.g. the BASIC compiler and library routines) are not pedagogically intended. We also consider a student who <u>authors</u> a CAI program such as a simulation or a tutorial to be in SOLO mode. Finer distinctions within these modes can be found in [1].

Our general approach has been to encourage the orderly growth of a central computing "center" in the school (analogous to the library), in which are found terminals which access a single general-purpose time-sharing utility. Any one of these terminals can be used in any of the modes described, with students going as deeply into the system as their abilities, curiosity, and the curriculum material guidance we supply permit. Students and teachers also manage on-line and off-line file storage for the system.

It is our preliminary estimate that the capability of functioning in SOLO mode is elicited more readily than much of the

---

**The use of computers to support the administrative work of schools or vocational training for data-processing careers is not an immediate objective of our present project.

emphasis on drill, tutorial, and other dual mode CAI would seem to assume possible. Interestingly enough, however, the best route to "getting over the hump" into the world of SOLO mode computing for many students seems to be via such dual mode experience. After working with another person's tutorial or simulation (preferably his own teacher's), the student wants to know how it worked. It has become clear to us that this question should always be answerable, and within the context of the same system that generated the initial curiosity. Further, this first move into the world of SOLO mode computing should be preparatory for, and therefore compatible with, a later more sophisticated use of the same system for general purpose algorithmic computing (which, incidentally, some students are able to undertake almost immediately). We view this latter stage as very important, and continue to be amazed at the cognitive skills displayed by students functioning at this level. The present discussion will be limited, however, to a description of the system that makes possible and encourages the transitions between these various levels of computer usage.

3. SYSTEM REQUIREMENTS

There are three general requirements on a system within which such an educational approach will work:

a. Easy system access in all modes.

The simple, self-prompting system commands typical of most time-shared BASIC processors have proved to be understandable and usable by even nine and ten year olds. Because the use of such systems is easy and natural, students can put the bulk of their effort into

understanding the content of what they are doing. This is in contrast to one educational system we know of that required complex JCL commands. The students on this system never saw the forest for the trees.

If we are to allow a mix of dual and solo mode computing on a single system, with students and teachers continually making transitions between modes, the need for a simple uniform structure for controlling all features of the system is of even greater importance. This criterion should also apply to other activities on the system, such as file use, editing, and interactions with other processors.

b. <u>The full power of general purpose computing should be available to all users</u>.

Critics of educational technology worry about a "Hawthorne effect", pointing out that most innovation falls flat when the novelty wears thin. Exactly the reverse has been true in the world of <u>real</u> computing. Von Neumann's prognosis of the endless possibilities of truly general computing systems has been repeatedly verified. It is our opinion that the CAI systems of today and tomorrow should possess <u>all</u> of this potential.

To say this in another way, the elements in the human learning situation that ought to be most capitalized on in any innovative program are the internal resources of the learner himself. These come as standard equipment(!) and will work wonders if not stifled by artificial constraints. The surest way to avoid such inhibition in the use of technology is to have the unparalleled flexibility of

late generation computing systems always on tap for the learner and his mentors, with a crystal clear invitation to harness this power any time they deem appropriate.

c. The system should be "approachable" on the terms of the educator.

When we first worked with teachers in exploring the use of computers in education, we decided that they should understand what was going on, and that the best way to bring this about was to teach them all about "programming". The principle of insisting on understanding has proved to be correct, but the approach taken was wrong for all except a mathematically oriented minority. In deciding that the know-how of the computer scientist be superimposed on the educational world we were making a mistake analogous to an ill-conceived foreign-aid program. A little thought shows that it is the school-world culture that should dominate; its conventions and methods should be the starting point. Innovation added in this manner stands a much better chance of surviving the inevitable initial difficulties associated with change.

As will be seen below, the concept of DUAL mode computing, with teachers (and eventually their students) acting as authors has proved to be a good basis for entry into the total world of computing for most educators. In retrospect, we find ourselves amazed at not having realized that a teacher would assimilate the complex world of computing in terms of the control that is needed in good tutoring much more readily than from, say, tracing the iterations in a second order root finder! What we could not have foreseen is the

ingenious way in which teacher-organized groups of students continue to extend their work. As a result of these continual improvements, the problem of obsolesence of DUAL type programs because of exposure to large student populations is no longer one of our concerns.

4. THE NEWBASIC/CATALYST SYSTEM

Our current working version of a system that meets these criteria evolved from a prototype developed at the University of Pittsburgh by interfacing a CAI processor called CATALYST [2,3] with PIL [4], an interpretive language based on JOSS. A considerably improved system based on this experience has been implemented by Com-Share Inc. of Ann Arbor for use in our large school tests. The algorithmic language employed is a powerful extension of BASIC called NEWBASIC or NBS. In addition to all the niceties one might expect in a modern BASIC (full string manipulation, extensive function availability, direct mode, multiple statements, picture formats, full Boolean control, multiple data types, suffixes such as "if", "unless", "while", "until", and "for", etc.) NBS has two other distinctive features.

The first involves user defined functions which provide unlimited extensibility of the language. Section 5 will illustrate the application of this idea to fitting the language to the needs of non-numeric tutorial writing.

The second new feature is one that is an important first step in making the entire system user-interactive, rather than just

certain aspects of the NBS processor. For example, a user in NBS can go into and out of the editor, using its full power, without losing his place in what he was doing in NBS. Using the editor is no longer a confusing operation; its as simple as picking up an eraser when the need dictates. Similarly, appropriate executive system functions (e.g. looking at a directory of files) can be interspersed with a primary activity such as creating or running NBS programs, again without losing place.

From the pedagogical point of view, by far the most important example of this second concept occurs when a student, who is interacting (in DUAL mode) with someone elses tutorial or simulation program (written in NBS), is allowed to go into his own NBS (where he proceeds to function in SOLO mode) and then return to the original interaction without missing a beat. Simple examples of this will be given in Section 6; we have found the power of this feature to be a significant break-through in changing the attitude of many educators toward CAI.

5. EXTENDING NBS TO FACILITATE CAI AUTHORSHIP

Insofar as any BASIC implementation provides general algorithmic capability, it can be used to author dual mode programs, that is, programs which assist a user other than the author in achieving some educational goal. For many subjects the language constructs are really inadequate however, and fail to meet our third criterion. To illustrate some of the difficulties and how they can be overcome, we will examine one of the author guidance forms we have suc-

cessfully used to coordinate the efforts of a beginning group of tutorial writers (frequently a team of students under the leadership of their teacher).

The team selects a sequence of sub-areas A, B, C,... leading to the terminal goal decided upon. Each member of the team codes the program for one of these areas. Figure 1 shows an example of a strategy that might be employed by a team member in writing a tutorial exploring sub-area A. Since he uses line numbers 1000 to 1999, his work will mesh with the student exploring sub-area B who has line numbers 2000-2999, and so on.

(Insert Figure 1 Here)

Despite the relatively complex branching involved in this scheme, we find that teachers and students have no difficulty at all in working at this conceptual level right from the start (compare with the usual "find the sum of the first 100 integers" example in beginning programming courses). They also have no problem in coding such a scheme in BASIC since the suggested ranges of line numbers preclude clashes. The difficulties that do arise come from the traditions associated with algorithmic languages, and are of three kinds:

(1) Annoyance type constraints.

Creating the blocks of text symbolized by the rectangular boxes shown in Figure 1, or supplying a variety of reinforcement messages, come under this heading. NBS allows for several ways around the repetitious use of PRINT "..." to handle this. For example

```
1000 PR. "_____(text)_____     (LF)
         _____(text)_____     (LF)
         _____(text)_____"    (CR)
```

PRESENTATION AND SCAN 1000-1199 | EXPECTED RESPONSE 1200-1399 | EXPECTED ERROR #1 1400-1499 and 1500-1599

100 General explanation, initialize counts

1000 1090 Section A Explanation Question 1 on A

1100 ?

1110 Scan for Error #1

NO

1120 Scan for Error #2

NO

1130 Scan for Expected R

NO

1140 Set counters,scores

1150 Test criterion for continuing — FAIL → GOTO 9900

PASS

1160 1180 "SORRY--YOUR RESPONSE NOT UNDERSTOOD" Rephrase Explanation "PLEASE ANSWER AGAIN" Rephrase of Quest. 1

GOTO 1100

Note: student only sees material in rectangular boxes like this.

All other steps are for control.

(REIN)

1200 1220 Question 2 on A (to verify understanding)

1230 ?

1240 Scan for Expected R

NO

1250 Set counters,scores

1260 Test — FAIL → GOTO 9900

PASS

1270 1290 "NO"--Explain Expected Response "NOW LET'S RUN THROUGH THIS SECTION AGAIN"

1299 GOTO 1000

(RRIN)

1400 Set cntrs,s

1410 Test — FAIL → GOTO 9900

PASS

1420 1435 "NO..." Sub-explana. Sub-question

1440 ?

1442 Scan Exp. R

NO

1444 Set cntrs,s

1448 Test — FAIL → GOTO 9900

PASS

1450 1480 "NO!"-Rephrase sub-explanatn. Give hint, ask sub-question.

1490 GOTO 1440

(REIN)

1500 1530 "NOW LET'S GO BACK TO ORIG. PROBLEM" Restate Q. 1.

1540 GOTO 1100

9900

"PLEASE LOG OUT AND TAKE THIS PAPER TO ...

9999 END

1350 Print Score → 1360 Reset ctrs,sc → 1370 GOTO 2000 (Section B)

FIGURE 1

is a legal construct. An alternate manner of creating blocks of text is to precede and follow any text with the statements "FRAME" and "END OF FRAME". Reinforcement messages sequentially selected from two circular lists REIN and RRIN (really reinforce!) are supplied upon calls to these lists within scan statements (see line 200 in the example below).

(2) Difficulties that confound the non-programmer.

The initialization and incrementing of counters or scores, and the testing of these quantities to effect branching turn out to be confusing ideas for people not brought up in the algorithmic tradition. The use of "FOR" loops causes even more consternation. These difficulties are not permanent, and the ways of the world of computing are eventually found to be valuable. To bridge the gap, however, we find the use of a new "filter" command of the form

```
IF PASS > 3  THEN 850
```

(which might preferably be read as "PASS 3 TIMES THEN GO TO 850") of great value. The command does just what it says, and provides exactly the kind of looping control that is familiar territory for a teacher. Once it is in use, we find it very easy to introduce incrementing via assignments of the form LET S = S - 10, with full appreciation of the consequences of this "strange" operation.

Other uses of the PASS command are possible for fancier control, and several versions of a REPEAT statement will provide for additional looping control.

(3) Difficulties that confound programmers.

While we don't try to encourage too much "cuteness" in tutorial programs (although student authored programs can capitalize on local humor), we find that people are quite a bit more creative when responses in free form are permitted. This implies the ability to scan a response in a fairly general manner. To accomplish this, the first version of NBS/CATALYST permitted statements of the kind

```
100 INPUT R$        (R$ thus holds the response typed by the student)
200 IF R$ CON(TAINS) (THE) WO(RD) "MARINE" AND "PRESSURE" BUT NOT
    "FORCE" REIN, (THEN) GO TO 3510
```

The "WORD" modifier says we accept the string "MARINE" only if it is not part of another word. Thus "SUBMARINE" would not be accepted. The rest of the syntax is self explanatory.

These statements were handled by being converted (through a preprocessor) to legal NBS statements of the form

```
IF ICO (R$,"MARINE",1) AND ICO (R$,"PRESSURE",0) BUT NOT ICO (R$,
    "RPM",0) CALL REIN, GO TO 3510
```

The preceding makes sense in NBS because of the ability to define functions like ICO or subroutines like REIN. Our high school students accidentally discovered the functions, figured out how the third parameter acts to strip off special symbols before deciding what a "word" is, and proceeded to always use the function call form. Although our staff never would have contemplated giving

a group of beginners a lecture on "the use of implicit function calls with Boolean return values", it turned out to be a concept that was easily assimilated in the proper context. We suspect that we can therefore continue to use the second form for many users.

Examples of two other useful scan routines are:

(1) IS (R$,A$,1)

"IS" returns the value 'true' when R$ matches A$ as a word. A little thought shows that the only Boolean connector that should really be used between two IS calls is XOR, although in practice OR has the same effect.

(2) IBEF(R$,A$,1,B$,0)

"IBEF" returns the value 'true' when the word A$ is found in R$ before the string B$, where 1 and 0 specify word and string respectively.

There is no limit to the number of such routines available in NBS. Normally we expect that an experienced programmer will write these functions according to the specifications of the users, although we recently had an eleven year old student write routines which provide complete cursor control for NBS on Datapoint 3300 terminals.

The preceding routines emphasize the handling of non-numeric responses. When numeric responses are requested, the normal NBS syntax handles things nicely. A frequent pattern used by our authors for such cases is the following:

(1) A problem is presented, but with data supplied by either a random number generator or the user

himself. In this way the drill or tutorial does not have "known" answers.

(2) Standard NBS statements calculate the correct answer and store the result in a variable, say A. (The user of this program doesn't see this happening.)

(3) The user is then asked for his answer, and his response is accepted if it agrees with the calculated answer within say 3 per cent.

One coding* for this last step might be

```
300 INPUT R
400 IF (R - .03*R) < A AND A < (R + .03*R) CALL REIN, GO TO 900
```

The random generator is also used to select general data to make CAI programs more interesting. For example, a program written by a geography teacher invites the student to "discover" a country randomly selected by the computer. The student asks questions based on a map he has in order to narrow down the country before making his guess. A selective scoring system which the student sees after each of his inputs (questions or guesses) encourages good questions. Most students end up with a thorough assimilation of the essential features of the map in short order, which of course is the educational goal.

---

*Deliberate inefficiencies are used in some of our codings to help beginners understand concepts; this also invites user modification of programs which is sound pedagogy.

6. MULTI-LEVEL BASIC FOR MIXING SOLO AND DUAL MODES

The feature of the NBS system that most excites the educators we have worked with is the option that a student has of not answering an input request made of him by a DUAL mode program until he has done a little SOLO mode exploration on his own. The student does this exploration at the same terminal used for the dual lesson, with no break in the flow of the overall session, and with the full power of NBS available to him. We tell students that the best thing that can happen as a result of such exploration is that they later contact their teachers with their hard copy demonstration of explorations not anticipated by the original author.

The best way to illustrate the use of this feature is by way of example. The illustrations are meant only to suggest applications, and are not intended to typify anything resembling good CAI programming. In each example we show a listing of the NBS tutorial program, followed by a student interaction with that program, where the student accesses his own NBS system by typing @NBS.

Example 1

LISTING:

```
>10 PR."THIS IS A TRIVIAL EXAMPLE OF A TUTORIAL WHERE
YOU MAY USE @NBS TO RETRIEVE DATA FROM THE FILE /RIPLEY/.
DO YOU HAVE YOUR FILE INSTRUCTION SHEET WITH YOU?"
>20 LET Y$=",YES,YUP,SURE,OF COURSE,AFFIRMATIVE,"
>30 INPUT R$
>40 IF IEQIV(R$,Y$,0) GOTO 70
>50 PR."PLEASE LOGOUT AND OBTAIN THE FILE INSTRUCTION
SHEET.  PRACTICE USING IT ON A TERMINAL BEFORE TRYING
```

```
THIS LESSON."
>60 STOP
>70 PR."HERE IS YOUR FIRST QUESTION.....
    NAME AN OBELISK FOUND IN AFRICA"
>80 INPUT R$
>90 IF ICO(R$,"KARNAK",1) CALL RRIN,GOTO 120
>100 PR."SORRY - YOUR ANSWER ISN'T ONE WE ANTICIPATED.
WE HAD THE 'TEMPLE OF KARNAK' IN MIND"
>120 PR."LET'S TRY ANOTHER QUESTION
          ......ETC....."
>130 END
```

AN INTERACTION:

```
>RUN /TRIVIAL/
THIS IS A TRIVIAL EXAMPLE OF A TUTORIAL WHERE
YOU MAY USE @NBS TO RETRIEVE DATA FROM THE FILE /RIPLEY/.
DO YOU HAVE YOUR FILE INSTRUCTION SHEET WITH YOU?
?YUP
HERE IS YOUR FIRST QUESTION.....
     NAME AN OBELISK FOUND IN AFRICA
?@NBS
VER. AUG 12 17:20
>OPEN /RIPLEY/ FOR INPUT 2
>INPUT FROM 2, A$(I) FOR I=1 TO 5
>PRINT A$(I) FOR I=1 TO 5
 4 ITEMS:
THE WASHINGTON MONUMENT
THE HONG KONG HILTON
THE TEMPLE OF KARNAK
THE HANGING GARDENS
```

```
>CLOSE 2
>EXIT
RESPOND TO LAST INPUT REQUEST
?THE TEMPLE OF KARNAK IS THE ANSWER
VERY GOOD INDEED!
LET'S TRY ANOTHER QUESTION
      ......ETC..........
```

Example 2

LISTING:

```
>100 PR."SLIDE RULE DRILL: ESTIMATING CUBE ROOTS"
>110 REM WE ASSUME USE OF A RANDOM GENERATOR AND A
>120 REM LINEAR TRF TO SUPPLY A VALUE FOR X IN LINE 130
>130 LET X=37595.4
>140 REM WE ASSUME THAT A SUBROUTINE WOULD BE CALLED  IN LINE
>150 REM 160 FOR CALCULATING ANSWERS TO MORE GENERAL PROBLEMS
>160 LET C=x↑.333333
>170 PR."PLEASE ESTIMATE THE CUBE ROOT OF":X
>180 INPUT R
>190 IF R>C-C*.05 AND R<C+C*.05 CALL REIN, GOTO 300
>200 IF R>=C+C*.05 GOTO 240
>210 IF R<=C-C*.05 GOTO 260
>220 PR."DO NOT UNDERSTAND - PLEASE REPEAT"
>230 GOTO 170
>240 PR."NO -- HINT: YOUR ESTIMATE IS TOO LARGE"
>250 GOTO 170
>260 PR."NO -- HINT: YOUR ESTIMATE IS TOO SMALL"
>270 GOTO 170
```

```
>300 PR."LET'S TRY ANOTHER - IF YOU WISH TO"
>310 PR."STOP AT ANY TIME PRESS THE 'ESC' KEY"
>320 GOTO 130
>330 END
```

AN INTERACTION:

```
>RUN /SLIDE5/
SLIDE RULE DRILL: ESTIMATING CUBE ROOTS
PLEASE ESTIMATE THE CUBE ROOT OF 37595.4
?10
NO -- HINT: YOUR ESTIMATE IS TOO SMALL
PLEASE ESTIMATE THE CUBE ROOT OF 37595.4
?60
NO -- HINT: YOUR ESTIMATE IS TOO LARGE
PLEASE ESTIMATE THE CUBE ROOT OF 37595.4
?@NBS
VER. AUG 12 17:20
>5 INPUT A,B
>10 FOR I=A TO B
>15 PRINT I;I*I*I
>20 NEXT I
>25 END
>RUN
?35 40
 35     42875
 36     46656
 37     50653
 38     54872
 39     59319
 40     64000
```

```
>RUN
?30 35
30      27000
31      29791
32      32768
33      35937
34      39304
35      42875
>PRINT 33.4*33.4*33.4
 37259.704
>PRINT 33.5*33.5*33.5
 37595.375
>EXIT
PLEASE RESPOND TO LAST INPUT REQUEST
?33.5
CORRECT
LET'S TRY ANOTHER - IF YOU WISH TO
STOP AT ANY TIME PRESS THE 'ESC' KEY
PLEASE ESTIMATE THE CUBE ROOT OF.....
?←←ESC:          180
```

7. SUMMARY

Although the emphasis in the preceding description of the NEWBASIC/CATALYST SYSTEM has been on its application to CAI, we feel that the true potential lies in the way the system invites users of all ages, talents, and backgrounds to tap the full power of computers for learning on an individually tailored basis. Some of our students are already making use of the fact that NBS can call on a very extensive library of FORTRAN routines. This really means that these students are looking in on much of the history of computing, but only as they are ready for each item. Thus students who a few months ago thought that generating factorials was 'higher math' are now using the Gamma function call with a pretty good appreciation of what this represents.

The fact that NBS resides in a general purpose time-sharing system is also being taken advantage of, with a few students beginning to explore everything from assembly language to general simulation languages. We are quite convinced that nothing less than the power and complexity of such a system is a proper match to the wondrous potential of the human learner, and recommend that educators give serious consideration to thinking at this level when investigating computer technology for their schools.